A Security Analysis of an In Vehicle Infotainment and App Platform

Sahar Mazloom[†]
George Mason University

Mohammad Rezaeirad[†]
George Mason University

Aaron Hunter
George Mason University

Damon McCoy
New York University

[†] *Lead co-authors contributed equally to this work*

Abstract

There is an increasing trend in the automotive industry towards integrating trusted third-party apps with In-Vehicle-Infotainment systems (IVI) via smartphones. This integration is typically facilitated by a pair of apps, one that executes on the smartphone and the other executes on the IVI which is connected to the Vehicle's Controller Area Network (CAN) bus. Throughout the evolution of these IVI and App platforms, there has been little public analysis of the security of these protocols and the frameworks that implement these apps on the IVI. This raises the question: to what extent are these apps, protocols and underlining IVI implementations vulnerable to an attacker who might gain control of a driver's smartphone?

In this paper, we focus on gaining insights into this question by performing a comprehensive security analysis on an IVI system that is included in at least one 2015 model vehicle from a major automotive manufacturer. This IVI system included vestigial support for the MirrorLink protocol which is intentionally disabled but can be enabled by updating a single configuration value after applying a publicly available firmware update that is securely signed by the manufacturer. Based on our analysis, we document and demonstrate insecurities in the MirrorLink protocol and IVI implementation that could potentially enable an attacker with control of a driver's smartphone to send malicious messages on the vehicle's internal network.

1 Introduction

Modern automobiles are increasingly controlled by computerized systems that are directly or indirectly connected to the Internet. This in turn, means that automobiles are potentially vulnerable to attack from adversaries who can gain control of one of these external devices. Previous works have demonstrated cars are vulnerable to an attacker that has internal access to the Vehicle's Controller Area Network (CAN) bus or can remotely connect to automotive systems [14, 18, 21, 11, 22, 25, 35].

Despite these risks, the number of external devices that are connected to both an automotive system and the Internet continues to increase. In particular, there is an increasing trend towards the integration of smartphones and In-Vehicle-Infotainment systems (IVI) connected to the CAN bus. This integration between IVI and smartphone is typically facilitated by a pair of apps, one that executes on the smartphone and the other executes on the IVI. Currently these app platforms are a closed ecosystem of trusted third-party developers, but many automotive manufactures plan to open up their IVI app platforms to a larger pool of less vetted app developers [27]. The initial app platforms, such as Toyota Entune [5], BMW ConnectedDrive [4], Cadillac CUE [2], Ford Sync AppLink [3], were largely proprietary to a single manufacturer and caused issues for developers who needed to support large numbers of protocols and legacy support. Recently a number of standardized protocols have emerged to enable seamless connectivity between smartphones and the car infotainment systems, such as Android Auto [7], Apple CarPlay [8], GENIVI [6], and MirrorLink [12].

Throughout the evolution of these IVI and app platforms, there has been little public analysis of the security of these protocols. This raises the question: to what extent are these apps, protocols and underlining IVI implementations vulnerable to an attacker who can gain control of a driver's smartphone? This paper focuses on gaining insight into this question by performing a methodical empirical analysis of the security of a single on the road IVI, along with its corresponding smartphone App, and communication protocol. Note that the MirrorLink code on this IVI is vestigial functionality that is disabled by default and there is no indication that the manufacturer intended to allow drivers to enable it. However, MirrorLink can be enabled on this IVI by changing a sin-

gle configuration value in a file. We also found evidence that some drivers have likely enabled this functionality and there are instructions online which document how to do so.

Motivation: Recent studies have demonstrated that internal and remote attacks against safety critical automotive systems are possible. However, this paper presents, to the best of our knowledge, the first comprehensive empirical security analysis of a smartphone to modern IVI app protocol included in at least one 2015 model vehicle from a major automotive manufacturer. This IVI system includes vestigial support for the MirrorLink protocol. Though this protocol is disabled by default, it can be enabled by changing a single configuration value after a publicly available firmware update that is securely signed by the manufacturer. As part of our analysis, we:

1. Document and perform a security analysis of the MirrorLink protocol standard and how the protocol is implemented.

2. Perform a security analysis of critical parts of the IVI MirrorLink app implementation that parse potentially malicious input from the smartphone using static and dynamic analysis.

3. Demonstrate the vulnerabilities in the IVI MirrorLink app implementation that can potentially lead to a malicious smartphone causing the IVI to send arbitrary CAN massages.

4. Discuss how these protocols and IVI firmware can be rearchitected to improve the security of these systems against potentially malicious smartphones.

2 Background

In this section, we provide background on the MirrorLink protocol and the IVI platform integrated into several 2015 and 2016 vehicles currently on the road. The MirrorLink code on this IVI is vestigial functionality that is disabled by default, but it can be enabled by changing a single configuration value. We found evidence that some drivers have enabled this functionality and step-by-step instructions are available online documenting how to enable MirrorLink. The IVI also includes several apps, such as Pandora streaming music and BringGo navigation, that use proprietary protocols. However, given that most automotive manufacturers are moving away from proprietary protocols and towards standardized protocols for future IVI app platforms, we focus on one of these, MirrorLink, in our study.

2.1 MirrorLink Protocol Specification

The MirrorLink standard[1] specifies a protocol that facilitates the integration of a smartphone to an automotive infotainment system. The main goal is to offer the user the ability to use her smartphone applications on the larger screen of the infotainment system instead of purchasing a more expensive IVI with built in Internet connectivity and applications. So in this case, for instance, she can use her smartphone navigation apps instead of paying extra for a navigation system tied to the car. Moreover, upgrading an infotainment system is not as easy as a smartphone. This paradigm is based on a client-server model in which the smartphone acts as a server and the infotainment system is its client. In this model, all of the applications will run natively on the smartphone, and the screen of the smartphone will be replicated on the infotainment screen. There are some user interaction methods implemented in the car system such as steering wheel control keys, rotating knobs, touch screen or even voice-activated control that enable the head unit to be able to send user requests to the smartphone. The MirrorLink server (smartphone) runs the requested applications natively and then mirrors the display of the application on the infotainment screen.

The MirrorLink specification defines connectivity layers as illustrated in Figure 1.

Physical and data link layer: MirrorLink requires at least USB 2.0, because it meets minimum bandwidth requirement and at the same time provides a charging solution for the mobile device. Among different USB profiles, CDC/NCM was chosen for USB networking. Support for WLAN and Bluetooth is optional.

Networking and Transport layer: IPv4 is specified for the networking layer with both UDP and TCP used as transport layer protocols. Both the client (IVI) and server (smartphone) must support DHCP for IPv4 addressing.

Session and Application layer: The MirrorLink application layer comprises of four basic session layer components and it utilizes either TCP or UDP sockets to interact with these components: (1) UPnP: is used for service negotiation, advertisement of MirrorLink enabled server devices and client profiles, as well as control of MirrorLink server based applications. It uses UDP for broadcasting and advertisement, and TCP for remote application control. UPnP is designed to allow networked devices to seamlessly discover each other using the Simple Service Discovery Protocol (SSDP) and establish useful network service connections using a variety of other pro-

[1] The MirrorLink standard started out as a research project under the name of Terminal Mode, which was originally proposed by Nokia [10]. The Car Connectivity Consortium (CCC) is made up of various auto and electronic manufacturers who have joined together to establish an industry standard for certifying apps and devices that are both safe and useful for drivers, called *MirrorLink*.

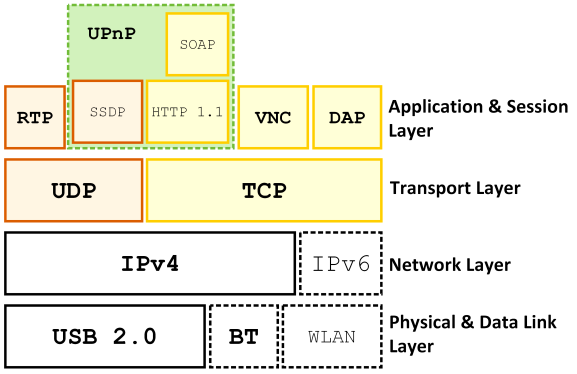

Figure 1: MirrorLink Connectivity Stack

tocols, such as the HTTP and Simple Object Access Protocol (SOAP). (2) VNC: provides the mirroring feature for MirrorLink protocol. It replicates the display's content of the MirrorLink server (smartphone) to the MirrorLink client (IVI). It transfers control information, such as key, pointer or touch events, as well as voice commands from client to server. TCP sockets are used in the VNC protocol. The content replication (mirroring) feature of MirrorLink is based on the VNC protocol, which is also used by many remote desktop applications. The VNC server runs on the mobile device and VNC client runs on the IVI system. When the user wants to run an application on the IVI system, a key or a pointer event is sent from the IVI (VNC client) to the mobile device as a command and control input. (3) RTP: Exchanges audio content (audio streaming) for different payload types is provided by RTP protocol. In addition to RTP, Bluetooth is also used for telephone applications (BT HFP) and media streaming (A2DP). (4) DAP: Device Attestation Protocol (DAP) [23] is responsible for verifying the hardware on which the MirrorLink server software operates on. This attestation mechanism is based on standard X.509 certificates, and refers to the MirrorLink client (IVI) verifying that the MirrorLink server (smartphone) is from a compliant manufacturer and running the approved software.

2.1.1 MirrorLink Security Mechanisms

Based on the MirrorLink specification, there are two main security mechanisms:
(1) Device Attestation Protocol (DAP), UPnP devices consider all other devices and users to be trustworthy, which means if an attacker can gain access to the local network she could potentially inject, alter or delete UPnP messages. In MirrorLink Protocol, DAP is facilitated by the UPnP protocol in order to confirm the compliance of the MirrorLink Server hardware and software. The attestation is based on standard X.509 certificates and the attestation mechanisms that are standardized by the Trusted Computing Group (TCG). This enables the MirrorLink server device to authenticate (or attest) itself to the MirrorLink client device using a manufacturer's certificate, which is included in the DAP message exchange. The MirrorLink client should have the CA's public key (in the MirrorLink context, Car Connectivity Consortium (CCC) is the certificate authority) to validate the MirrorLink server's certificate. The server devices private key(s) must be stored securely and this can be done via Hardware-based Mobile Trust Module (MTM) implementation [13] or equivalent [9], and storage on OS for which integrity has been verified with hardware-assisted secure boot process [29]. The attestation process starts with the MirrorLink client sending an `attestationRequest` message and the server replying with an`attestationResponse` message, both in XML format. The main goal of this process is to enable the client device to attest that the MirrorLink server device is a **genuine device** from an authorized manufacturer and also that its software components such as VNC, UPnP, and RTP are certified. The usefulness of DAP is to mitigate the threat of drivers connecting insecure devices to the IVI.

(2) Content Attestation, According to the MirrorLink specification, authentication and security is handled outside the VNC protocol by the link layer and transport layer. Therefore, the VNC client does not require the VNC server to offer additional security or authentication features. For this reason, the MirrorLink protocol specifies that the security type in the VNC handshaking phase is set to `None`. However, there is a pair of content attestation messages that allows a MirrorLink client to verify the received content from the server. The purpose of these messages is to prevent the man-in-the-middle attack, in which the frame-buffer contents or settings can be modified maliciously. The VNC client can initiate the challenge-response protocol at any time with the VNC server. The client starts the challenge by sending the `ContentAttestationRequest` before sending the next `FramebufferUpdateRequest`. The Server should reply back with a signed `ContentAttestationResponse` immediately after forwarding the `FramebufferResponse`. The signing algorithm can be defined in the `ContentAttestationRequest` message.

3 Related Work

Previous research [19] identified that request messages sent on an automotive CAN can be easily eavesdropped on and injected into the CAN bus. In 2010, Koscher et al. [22] more comprehensively demonstrated that an attacker with access to a 2008 Chevy Impala's internal

network (i.e. CAN) can override the driver's input and launch a number of attacks ranging from disabling and enabling the brakes, to turning the engine off and on. Miller and Valasek [25] subsequently replicated many of these attacks in 2013 on a 2010 Toyota Prius and Ford Escape. In addition, Hope et al. [20] presented an attack against a window controller. This previous work illustrates the systemic insecurity of internal automotive networks with the limitation that all these attacks require access to the vehicle's internal CAN bus.

Checkoway et al. [11] first demonstrated that remote and indirect vulnerabilities in automotive ECUs would enable an attacker to gain access to the internal network of a vehicle in 2011. These efforts included exposing remotely accessible vulnerabilities in the OnStar telematics unit analog modem connected to the cellular network and at short range via Bluetooth paired devices. In addition, this study crafted a WMA file that exploited a buffer overflow in the CD player's WMA parser that allowed for indirect physical access attacks. In the same vein, Miller and Valasek [26] later staged a remote attack over the cellular network by compromising a software vulnerability in the IVI. Foster et al. [14] crafted SMS messages to exploit a popular aftermarket hardware dongle that connects to a vehicle via the standard OBD-II port. Finally, Kamkar previously exposed security flaws in the GM OnStar app that enabled invoking API calls of this app without authentication [18]. This security flaw only enabled sending CAN messages that were predefined by the OnStar app and did not enable sending CAN messages that were not predefined by the OnStar app. Our analysis exposes weaknesses that can potentially lead to sending arbitrary CAN messages.

Previous work has either required direct access to the vehicle's internal CAN network or targeted a specific remotely accessible ECU that implements a proprietary protocol. In this paper, we perform a security assessment and expose remotely accessible security weaknesses in an IVI system that implements MirrorLink, one of several competing standardized smartphone to IVI protocols. Like similar protocols, MirrorLink in the process of being deployed by many major automotive manufacturers, and will be potentially deployed by other vehicle manufacturers and IVI suppliers[2]. These IVIs might also be the first component in vehicles that will execute apps from an open third-party developer ecosystem [31]. For this reason, we feel it is critical to assess the security of these devices and protocols before they are deployed more widely.

[2] http://www.mirrorlink.com/cars/

4 Attack Model

In evaluating the security of the vestigial MirrorLink implementation on our IVI, we assume that the attacker does not have direct physical access to the specific target IVI, but does have access to an IVI with the same hardware that runs the same firmware. We also assume that the attacker can use physical reverse engineering tools and methods (i.e. JTAG debugger, chip readers, and network sniffing devices) to craft an exploit that would be able to remotely compromise the target IVI. However, these same physical reverse engineering methods cannot be used by the attacker to directly compromise the target IVI.

We assume that the attackers can also install a malicious app onto a specific MirrorLink enabled smartphone that will be connected to the target IVI. Such an installation could be done either with an exploit or by socially engineering someone into installing the app. The phone would either need to be jailbroken or the malicious app would require a privilege escalation vulnerability to gain root access. Another less targeted attack vector might be for the attacker to install their malicious app on a large set of smartphones, some of which would likely be connected to an IVI that is MirrorLink enabled and matches the hardware and firmware that is vulnerable to the attacker's IVI exploit. Finally, we assume that the IVI is connected to the automobile's internal CAN and that the exploit can trigger the IVI to send malicious CAN messages that override the driver's input to safety critical ECUs. These CAN attacks could be similar to those demonstrated by several previous studies [22, 25].

5 Security Assessment

In this section we begin our security analysis by describing how we obtained copies of the files on the firmware and what useful debugging interfaces we discovered. We then discuss how we were able to capture traffic between the smartphone and IVI, and what was learned from our analysis of this captured traffic. Finally, we present the findings from our MirrorLink software security analysis.

In order to perform our security analysis, we bought an IVI from a 2015 vehicle on eBay. Our initial investigation showed that this IVI was not MirrorLink enabled. However, the IVI firmware is updatable by downloading and placing a validly signed update image onto a USB drive. We found a publicly available firmware update for this IVI that was cryptographically signed by the automotive manufacturer which included an initially disabled implementation of MirrorLink. We then followed step-by-step instructions that are publicly available online to change one configuration value that enabled MirrorLink. Since the update is validly signed by the auto-

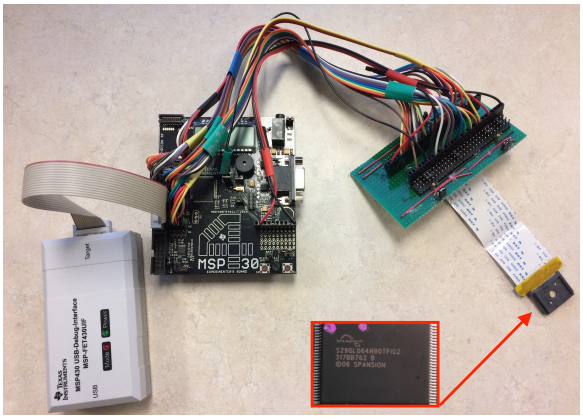

Figure 2: Flash reader is built with the aid of MSP430 controller and suitable socket for the target TSOP NOR flash chip size and pin number

motive manufacturer, we assume that this is an official firmware update that included vestigial MirrorLink functionality that some drivers might be enabling.

5.1 Preparation and Data Acquisition

This section describes our methods to: (1) gain access to and extract data stored on memory chips; (2) obtain application data, kernel and configuration files from these chips; (3) discover and utilize physically accessible pins in order to discover console/debugging interfaces. These pins enabled us to perform dynamic debugging and increased control over IVI.

5.1.1 Data Extraction

We identified a NOR flash chip utilized in this IVI. We started by building a flash reader to read NOR flash, as is shown in Figure 2.

The NOR flash chip contained the BIOS, a boot-loader and two root certificates, one from the MirrorLink consortium, and the other from the automotive manufacturer (see Figure 3). In order to install an unofficial firmware image, these certificates would need to be replaced with the ones used to sign the image. For this reason, it would not be feasible for a remote attacker to attempt to trick a driver into installing a potentially malicious image update, unless she gained access to the automotive manufacturer's private signing keys and generate a validly signed image update.

We then unpacked the update image and obtained the Windows embedded CE runtime image (`NK.BIN`), user application executables, kernel executables [3], configuration files, and a complete image of NOR flash.

[3]These also can be obtained by extracting the `NK.BIN` via `Bin*` tools as part of platform builder.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=                                    Code Signature CA
        Validity
            Not Before:
            Not After :
        Subject: O=                                   Code Signature CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:

                    <omitted>

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 12374219816782257553 (0xabba0f5ca7e7dd91)
    Signature Algorithm: sha512WithRSAEncryption
        Issuer: CN=CTS Root CA, O=Car Connectivity Consortium
        Validity
            Not Before: Oct  5 12:10:59 2012 GMT
            Not After : Oct  5 12:10:59 2032 GMT
        Subject: CN=CTS Root CA, O=Car Connectivity Consortium
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:

                    <omitted>
```

Figure 3: MirrorLink and automotive manufacture root cert extracted from NOR flash image

During our reverse engineering of these executables, we discovered a subroutine in `AppMain.exe` that enables Development Mode (`DevMode`). We found that the password is included in plain-text (clear) in the executable and there is a static comparison for the password required to enable DevMode mode. After entering this password on IVI, the Development Mode was enabled and several configuration options to assist developers in debugging the IVI became available, such as offering access to Windows CE GUI (`Explorer mode`). In this mode, we enabled the ActiveSync protocol, a CoreCon debugging connection that is used to establish a link between the development workstation and the CE device on the IVI by running an executable left on the system to enable debugging, as is shown in Listing 1.

Listing 1: UART debug messages showing that ActiveSync is enabled.

```
[AppLink] MgrTsk_ChangeDebugLevel: 2
wince_usbware_exit: USER MODE
USBware::UWD_Close: Started pContext 0xD0B6EB88
OEMInterruptDisable:: DDMA:maskc, inten:ff
[DRVMGR] Set ActiveSync Mode (1)
[DRVMGR] Open Success!!!!!
[DRVMGR] enable USB Host & OTG phy(NEW TYPE)
```

5.1.2 Interface Discovery

During our analysis we also discovered two interfaces useful for debugging purposes.

JTAG: During our hardware analysis, we detected a JTAG (Joint Test Action Group) interface via ac-

tive probing with a JTAGulator [15]. The target SoC (AU1340) does not support JTAG censorship/protection. We used a JTAG debugger (Lauterbach Power Debug [1]) to access chips of IVI. This JTAG port enabled us to access all the addressable memory of the system as well as CPU registers. The Lauterbach JTAG debugger is WinCE-aware and GDB enabled, which enabled us to connect it to an IDA Pro disassembler. In turn, this configuration allowed us to perform dynamic analysis by attaching to the processes on the IVI, setting breakpoints, reading process memories, and inspecting other properties of the IVI.

UART: We discovered UART (Universal Asynchronous Receiver/Transmitter) ports via active probing (JTAGulating). With the help of the SoC data sheet, we also traced the discovered test points (`RX`, `TX`) to actual CPU pins. This was done to make sure there was no fuse or resistor in place to block the `RX` or `TX`. However, after multiple tests, we found the RX is disabled (presentation only). The immediate result of UART debugging was the discovery of boot-sequence messages. We used other debug messages in development of our demonstration malicious app.

ActiveSync: Embedded CE Platform Builder provides a collection of remote tools to help with the debugging process. With the help of these tools we accessed the OS runtime image running on IVI remotely from a workstation. We employed the *Remote Heap Walker* tool to examine heap layout and memory contents for selected process running on the IVI.

5.2 USB Traffic Analysis

In Section 2, we provided a brief overview of the MirrorLink protocol specification. This section describes an empirical investigation of the specific MirrorLink protocol implementation on our IVI. We want to highlight how implementation of the components and parameters can ensure proper connectivity and secure smartphone communication to the IVI system. In order to investigate the protocol behavior, we first needed to monitor and capture the USB traffic between the smartphone and the IVI. Afterwards, we conducted analysis on the captured traffic. For our investigation, we utilized a Samsung Galaxy SIII smartphone running Jelly Bean version 4.1.1. We installed DriveLink application version 1.1.027 [4] on the smartphone. Finally, we used the IVI system that was salvaged from a 2015 automobile, which supports MirrorLink version 1.0.1 after updating the firmware and enabling it.

To capture USB traffic, we built the USB sniffer shown in Figure 4 using the BeagleBoard-xM [5] and a variant of an open source USB packet capture module produced by a Google Summer of Code project [28]. More information on how to build and use this USB Sniffer can be found at [24].

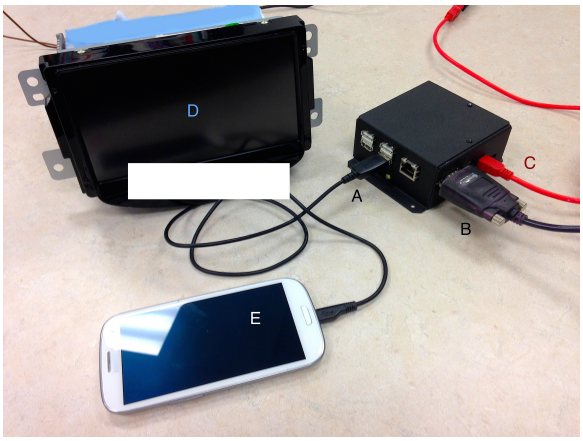

Figure 4: USB Sniffer setup using Beagleboard XM. "C" is the USB host port that is converted to a USB device port letting Beagleboard be accepted by Infotainment unit "D" (USB port on Infotainment is a USB host). "A" is a USB host that serves the Smartphone "E". "B" is a serial console port on the Beagleboard.

5.2.1 Observations

Based on our analysis of captured USB traffic between our smartphone and IVI, we can describe here our observations and understanding of the MirrorLink protocol. The initial phase of the protocol is the link-layer configuration phase in which the USB class is negotiated and accepted, resulting in both the smartphone and IVI USB interfaces being correctly configured. Next, the smartphone acts as a DHCP server and offers an IP address (`192.168.42.242`) to the client (IVI). After this step, the UPnP advertisement and negotiation messages, which are XML configuration files as shown in Figure 5, are being sent. The UPnP session is started by the smartphone, advertising its services and pointing at a URL, from which the IVI can download the `description.xml`. This XML file includes additional URLs for `tmclient.xml`, `tmapplicationserver.xml`, that are subsequently downloaded by the IVI. In the other direction, the IVI utilizes a SOAP over HTTP protocol to communicate service controlling and eventing messages back and forth with the smartphone. These service messages are: (1) IVI sends and confirms its hardware and software configuration to the smartphone, (2) IVI retrieves a list of avail-

[4] `https://play.google.com/store/apps/details?id=app.scm`

[5] `http://beagleboard.org/beagleboard-xm`

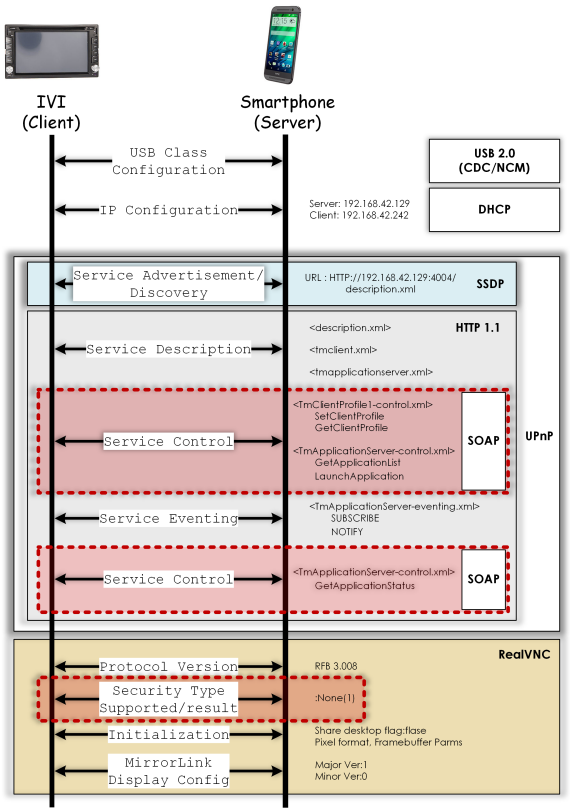

Figure 5: MirrorLink Protocol Stack

able application(s) (e.g, RTP, Bluetooth A2DP), (3) that enables the IVI to launch these available applications, (4) and to communicate application status with IVI.

From the captured packet traces, we found no indication of key exchange, encryption or authentication for these XMLs. None of these XMLs were signed; therefore if the security model of the MirrorLink protocol which largely relies on the security of the link-layer, is invalidated by an attacker who controls a device with access to the local link-layer segment, an attacker can control input to the IVI system. If the IVI system is vulnerable to attack from a malicious mobile device, the IVI might be used as a bridge onto the car's internal CAN network.

The DAP [6] protocol was not used by the IVI to restrict connections to only a set of trusted smartphone manufacturers. This means that any device or smartphone can connect to the IVI and potentially attempt to attack it.

After the MirrorLink application is launched, requests can be sent to the smartphone. The IVI acts as a VNC viewer client connecting to the VNC server (smartphone). Both the server and client negotiate and agree on `RFB 3.008` as protocol version. The server sends a list of security types that it could support (Security type values can only be 0 `[failed]`, 1 `[none]`, 2 `[VNC Authentication]`, 16 `[Tight Security Type]` or 19 `[VeNCrypt]`). Client selects type **1** `[none]`. This could enable an attacker to hijack the VNC session if they can gain access to the smartphone. In the captured traffic we have collected, there were no Content Attestation Request or Response messages.

[6] There is no evidence in the captured traffic that DAP is used. Since the purpose of DAP is to verify that the mobile device is from a compliant manufacturer, this could be used to mitigate the threat of insecure devices connecting to the IVI. However, in the case of an attacker gaining access to a genuine device, DAP will not mitigate these threats.

5.3 Software Analysis

As we mentioned earlier in this section, we managed to extract data from a USB update image and NOR flash. Here, we describe our security analysis of selected MirrorLink and CAN controller related binaries. To examine the presence of standard security protections (e.g, stack and heap protections), and to find out potential exploitable vulnerabilities, we performed static and dynamic analyses.

5.3.1 Static Analysis Results

Table 1 outlines the memory protection mechanisms [36, 34] used in compiling selected binaries for MirrorLink*, the CAN controller API°, which contains subroutines to interact with the Micom CAN controller, or both®.

Table 1: Memory protection mechanisms used in compiling selected binaries. `SafeSEH` and `DEP` are FALSE for all binaries because the used compiler does not support `SafeSEH` for the target architecture and also the target SoC does not support `DEP`.

Binary Name	ASLR	Stack Cookies
`AppLink.exe`®	TRUE	TRUE
`AppMain.exe`°	TRUE	TRUE
`ML_CERTIFICATION.dll`*	TRUE	TRUE
`CmnDll.dll`®	TRUE	TRUE
`MgrMcm.exe`°	TRUE	TRUE
`MgrSys.exe`°	TRUE	TRUE
`MgrVid.exe`°	FALSE	TRUE
`AppTM.exe`*	TRUE	TRUE
`TMScontrolPoint.dll`*	FALSE	TRUE

Based on our analysis, we have identified functions `Send2Micom_<redacted>Msg(union <redacted>_tx_msg_data_type const *, unsigned char, int)` and `SendMsg(union <redacted>_tx_msg_data_type const *, unsigned char, int)` that can accept data bytes of a CAN message and send this message on the CAN bus via the Micom CAN controller of the IVI.

The Micom CAN controller included in the IVI, is a `Renesas V850ES/SG3`. This is the same CAN con-

troller that was reprogrammed as part of a vulnerability in the Uconnect discovered by Miller and Valasek [26]. It includes a list of CAN IDs that limits the IVI to communicate with other devices using CAN bus. However, Miller and Valasek demonstrated a method of modifying the list of CAN IDs by updating that part of the Micom firmware [26].

Using static binary analysis, we discovered that the Micom CAN controller firmware can be updated using one of two different methods. The first is, by executing the `MgrUpg.exe` binary, which subsequently calls a function within `MgrUpg2.exe` that does the actual updating of the Micom CAN controller firmware. This binary is executed as part of the IVI main firmware update procedure. The Micom firmware can also be directly updated. This direct update is initiated via the DevMode interface and by copying unpacked Micom update files to a USB drive in a directory name of `\MicomeUpdate`. This second method bypasses the authentication of the firmware performed by the main IVI firmware update procedure.

We used the second method to update the Micom firmware, but with a slightly modified version of the original firmware. Our goal was to understand if there were any authentication or verification checks that would prevent the Micom firmware from being updated. We could not find any evidence of authentication and verification functions, although we did find a debug message ``start verify certificationWait reading'' in one of the subroutines of `MgrUpg2.exe` as well as UART debug messages. As best as we can tell, no actual firmware verification was performed. However, our IVI became locked and displayed the message "NO VIN" on the screen. This is likely due to a combination of an anti-theft method triggered by resetting a value in the IVI, and the fact that our IVI was not connected to an actual vehicle that broadcasts its VIN. This prevented any further experiments, but as future work we plan to understand how to unlock the IVI by sending the correct VIN over the CAN bus and explore if it is possible to modify the list of CAN IDs by updating that part of the Micom firmware.

The results of our Micom firmware reverse engineering efforts suggest that an attacker can potentially update the Micom firmware using methods similar to those used in a previous study [26]. After this, an attacker could potentially use these CAN functions to send arbitrary messages on the CAN bus if she is able to do one of two things: gain control flow of a process on the IVI and invoke this function with malicious inputs, or modify the input parameters to an existing call to one of these CAN functions.

In addition, during our static analysis of the OEM IVI developed MirrorLink binaries and DLLs on the IVI, we found that they were written in C++. We also found multiple instances of dangerous libc function [32] calls used to process potentially malicious input from the smartphone without proper boundary checking. For instance, in the XML parser function (`UPnPProcessAppList`) in `TMScontrolPoint.dll` there are multiple unbounded `memcpy` calls that are copying element values from the XML provided by the smartphone.

5.3.2 Dynamic Analysis Results

From our previous observations, we knew that there is no bounds checking or other protection mechanism. Given this, we constructed a demonstration malicious app that sends crafted XMLs to the IVI in order to understand if this would generate exception errors. To setup our dynamic analysis, we reverse engineered the MirrorLink app server (smartphone) part of the MirrorLink protocol. Then, we reimplemented part of the app that enables Ethernet over USB. With this in place, we could start mimicking the rest of the MirrorLink protocol by replicating and sending the correct UPnP messages. We crafted a replacement `description.xml` and `tmapplicationserver.xml` by changing some of the values of elements (e.g, `appname`, `appid`, `bluetoothaddress`) to overflow the static-sized struct buffers to which they would be copied as they were parsed.

Over the course of our experiments with our specially crafted XMLs, we witnessed multiple runtime exception errors. One such example is shown in Figure 6. We identified multiple heap memory corruption vulnerabilities that could be potentially used for exploitation purposes. In order to verify our results, we ran multiple experiments where we monitored the UART debug messages, heap content in heap-walker, and process memory in IDA Pro. Based on these experiments, we were able to verify that our specially crafted XMLs were overflowing buffers and causing heap memory corruptions. In turn, these memory corruptions were overwriting pointers to data structures and function pointers that resided on the heap.

6 Demonstration of Malicious Smartphone Application

To follow-up on our discovery of these heap overflows, and better understand if they could be used to gain execution control flow of the IVI, we developed a proof of concept malicious smartphone app. Figure 7 shows the GUI of our app and the IVI displaying an app where the `<appName>` XML element value length has been increased to overflow the heap.

```
Exception 'Access Violation' (2): Thread-Id=061300a6(
pth=853039c8), Proc-Id=052e001e(pprc=87b8f540) 'AppTm.exe',
VM-active=052e001e(pprc=87b8f540) 'AppTm.exe'
PC=421d15dc(tmscontrolpoint.dll+0x000115dc) RA=421d168c(
tmscontrolpoint.dll+0x0001168c) SP=0020f620, BVA=4848484c
ShowErr is running (ExcpCode : c0000005 / ExcpAddr : 421d15dc)
[ShowErr] MgrTsk (0x70021b40)!!!!!
Exception 'Alignment Error' (4): Thread-Id=06ea0006(
pth=85294978), Proc-Id=05940026(pprc=87b59440) 'AppTm.exe',
VM-active=05940026(pprc=87b59440) 'AppTm.exe'
PC=421c41c8(tmscontrolpoint.dll+0x000041c8) RA=421c41d0(
tmscontrolpoint.dll+0x000041d0) SP=001ff5e0, BVA=00000000
ShowErr is running (ExcpCode : 80000002 / ExcpAddr : 421c41c8)
[ShowErr] MgrTsk (0x70021b40)!!!!!
OEMInterruptDisable:: DDMA:maskc, inten:ff
[DRVMGR] BT reset
OEMInterruptDisable:: DDMA:maskc, inten:ff
```

Figure 6: UART debug messages showing multiple exception errors, program counter(PC), return address (RA), stack pointer (SP), binary name, and error type.

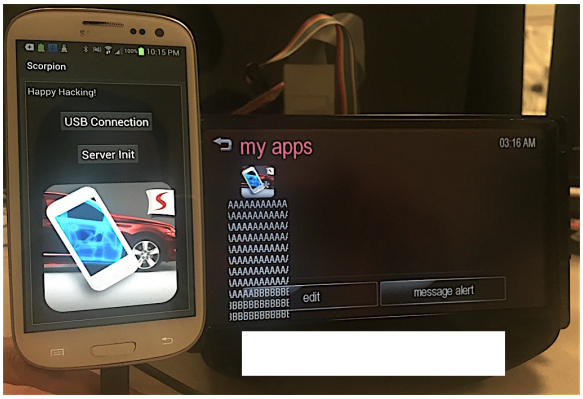

Figure 7: Malicious Smartphone Application: MirrorLink icon and name sent to the IVI

This app first correctly configures the USB mode and establishes network connectivity with the IVI, then correctly mimics the DHCP protocol and the initial discovery messages of the real DriveLink app between the smartphone and IVI. Once the IVI fetches the `description.xml` file, our app sends back a carefully crafted malicious version that includes several large values in elements of the XML scheme. These large values cause a series of heap overflows in `TMScontrolPoint.dll` that correctly arranges the heap memory and overwrites a function pointer on the heap. We also create and write a set of gadgets [30] to the heap that enable us to gain execution control flow, and reserve space on the heap that can be used to inject malicious code.

Since Windows Embedded CE 6 separates the heap meta-data from the actual heap, we must carefully overwrite a number of data pointers between the start of the heap and the target function pointer. If these data pointers are not overwritten with valid addresses, the process will crash before calling the overwritten function pointer on the heap. For demonstration purposes, our app modifies a debug message that is outputted to the UART. However, it is highly likely that an attacker could build a set of gadgets that executes a call to one of the CAN sending functions, to send a malicious massage onto the CAN bus. The UART output that our demonstration app generates is shown in Figure 8.

```
################################### UPnPInvoke_TmApplicationServer_LaunchApplicat
on_Sink ########################################
[MLCP] UPnPInvoke_TmApplicationServer_You Are Hacked!!!!!!!! Smile :)
 call UPnPRelease - 25
: 88 ms
[MLCP] UPnPInvoke_TmApplicationServer_You Are Hacked!!!!!!!! Smile :)
 call UPnPRelease - 25
: check 1
[MLCP] UPnPInvoke_TmApplicationServer_You Are Hacked!!!!!!!! Smile :)
 call UPnPRelease - 25
: check 2
[MLCP] UPnPInvoke_TmApplicationServer_You Are Hacked!!!!!!!! Smile :)
 call UPnPRelease - 25
: check 3 (1)
[MLCP] UPnPResponseSink_TmApplicationServer_LaunchApplication:  AppURI is null
[MLCP] UPnPResponseSink_TmApplicationServer_LaunchApplication: uri is null
[AppTM] UPnPResponseSink_TmApplicationServer_LaunchApplication:
```

Figure 8: The UART output generated by our demonstration malicious app.

7 Discussion

Our findings demonstrate vulnerabilities that are similar to those found in previous security studies of automotive systems [11, 25]. However, they are targeted towards what is potentially the more exposed surface area of an IVI and app platform. In this section, we will place our results into the context of the specific security challenges of IVI and app platforms, discuss some potential short and longer-term mitigation strategies, and address some of the future features on automotive manufacturer's road maps that will effect this ecosystem.

Security of MirrorLink protocol. Our analysis of the MirrorLink Protocol shows that few security features are specified at the application layer and that the designers of this protocol rely on the security of the link-layer protocol to protect the MirrorLink against attacks. The main security mechanism included in the specification is the Device Attestation Protocol (DAP) which is designed to prevent unauthorized hardware from accessing the IVI. The current MirrorLink protocol does not include any secure device pairing method. However, given our threat model, neither of these defenses would impede an attacker who can compromise a driver's smartphone that is likely already paired to the IVI and authorized hardware. Thankfully, the dangers posed by an attacker that can only invoke API calls exposed by apps on the IVI are limited at this point to relatively benign attacks, such as streaming unwanted music over the IVI. The worst case might be altering navigation directions. Given that the attacks are not devastating, the lack of security in the MirrorLink protocol is not ideal, but might be acceptable at present, and can be improved in later versions before more critical APIs are added to apps. The largest current

threat is if an attacker can gain more unfettered access to the IVI and CAN controller. Such an attack could permit an attacker to send arbitrary messages over the vehicle's CAN bus that could potentially effect safety critical systems.

IVI app containment. Our analysis of the firmware from the IVI shows that the MirrorLink client, which can be enabled on the IVI is written in a memory unsafe language, C++, and executed with administrator privileges on the bare-metal WinCE OS. This means that an attacker who can communicate with the IVI via a compromised smartphone can likely leverage one or more of the identified heap overflow vulnerabilities to gain control flow of the MirrorLink client. Our analysis of the CAN DLL shows that it likely provides a wide API interface to the CAN controller that would allow any privileged process on the IVI to load this DLL and send arbitrary and potentially malicious messages on the vehicle's CAN bus. This all points to the possibility that an attacker able to discover one of these vulnerabilities can craft an exploit to send out CAN malicious messages.

A short-term solution to this problem might be to modify the CAN controller firmware. A narrower interface that only permits sending predefined CAN messages needed for IVI operations would reduce this risk. It would also be a good idea to run processes, such as the MirrorLink process at a lower privilege level that does not grant them access to the CAN controller. The longer-term solution is to architect a virtualization and isolation model similar to that of Android or iOS, but adapted to the automotive domains [33]. If third-party app ecosystems emerge, then permission models will need to be developed for IVIs.

Developer practices. Our analysis of the MirrorLink executable showed that developers did make use of default protections provided by WinCE and the Visual Studio compiler, such as stack cookies and separation of heap meta-data from dynamic heap objects. Other security mechanisms, such as ASLR were enabled on some executables, but not for all of the MirrorLink executables. In addition, we found that developers commonly used memory unsafe versions of functions, such as memcpy and strcpy, when processing external inputs. It is difficult for most automotive manufacturers to perform security audits of firmware from suppliers, since normally manufacturers are not provided access to source code. In this development model suppliers, such as the supplier of the IVI in this instance, must improve their security audits of safety critical code or move away from lower-level memory unsafe programming languages to higher level languages that are more memory safe.

Future functionality. Currently MirrorLink is not enabled by default and requires a physical USB connection. However, the IVI hardware does include an 802.11 adapter and there are potential plans for MirrorLink to support WiFi as an alternate link-layer protocol. If this does become part of the standard, it enlarges the attack surface for remote attackers within a few 100 feet of the target vehicle, or those who can gain control of an 802.11 enabled device close to the vehicle. The other major addition to IVI system is the possibility of opening up the app developer ecosystem to less vetted third-party app developers. This would also enlarge the attack surface and hopefully necessitate implementation of stronger isolation of IVI apps.

8 Disclosure

As part of releasing our findings, we have chosen a responsible disclosure plan that includes an initial private disclosure, delayed publication, and left out critical details. We disclosed our findings of heap overflow vulnerabilities (but not the execution control flow exploit, since we had not implemented it yet) to the automotive manufacturer more than seven months ago. We have chosen not to name the manufacturer or supplier of the IVI, since we feel this is a systemic problem that likely effects other manufacturers and suppliers of app enabled IVI platforms. This will also provide additional time for the supplier to patch the vulnerabilities we found in our analysis. In addition, we have chosen not to disclose the full details of the vulnerabilities we have found given the difficulty of patching automotive systems [16, 17, 37].

Our goal in publishing about these security issues is to bring to light the current insecurities of these IVI and app platforms before they become widely deployed. Given our goal, we want to balance the possible gains of publication, in terms of improved security for IVI and app platforms with the risk of an attacker replicating the vulnerabilities we have found during our security analysis.

9 Conclusion

In this paper, we have presented the first security analysis of a smartphone to IVI protocol and implementation. Our security analyses show weaknesses in the MirrorLink protocol and more importantly, in the IVI installed in at least one 2015 model released by a major automotive manufacturer. The MirrorLink functionally implemented by this IVI was not enabled by default and there is no evidence that the manufacturer intended to allow drivers to enable it. However, there are publicly available instructions on how to enable it and some evidence that at least a few drivers have enabled it on their vehicles.

As a proof of concept, we have created a demonstration malicious app that exploits heap overflow vulnerabilities discovered in the implementation of MirrorLink

on the IVI. This vulnerability can allow attackers to gain control flow of a privileged process executing on the IVI. It is highly likely that these same vulnerabilities can be exploited by an attacker with control of a driver's smartphone, to potentially send malicious messages on the vehicle's internal CAN bus. Given our findings, we discussed how the security of these IVI app platforms can be made more resilient to these types of attacks. Our hope is that, this analysis will help motivate and spur more secure designs and implementations of smartphone to IVI platforms.

Acknowledgments

The authors thank Chris Everett, Houman Homayoun, Ubaidul Khan, Karl Koscher, and Parnian Najafi Borazjani for their assistance and feedback. Additionally, we thank the anonymous reviewers for their helpful feedback.

This work was supported in part by a contract from General Motors, the Department of Homeland Security under contract D15PC00239, and the National Science Foundation under contract 1205453. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of any funding sponsor.

References

[1] Power debug interface usb 3.0. http://www.lauterbach.com/frames.html?home.html.

[2] Cadillac cue. http://www.cadillac.com/cadillac-cue.html, 2010.

[3] Ford sync. http://owner.ford.com/how-tos.html#?tabCategory=sync, 2010.

[4] Bmw connected drive. http://www.bmw.com/com/en/insights/technology/connecteddrive/2013/, 2011.

[5] Toyota entune. http://www.toyota.com/entune/, 2011.

[6] Genivi. http://www.genivi.org/, 2012.

[7] Android auto. https://www.android.com/auto/, 2014.

[8] Apple carplay. http://www.apple.com/ios/carplay/, 2014.

[9] ARM, A. Security technology building a secure system using trustzone technology (white paper). *ARM Limited* (2009).

[10] BOSE, R., BRAKENSIEK, J., AND PARK, K.-Y. Terminal mode: transforming mobile devices into automotive application platforms. In *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications* (2010), ACM, pp. 148–155.

[11] CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., SAVAGE, S., KOSCHER, K., CZESKIS, A., ROESNER, F., AND KOHNO, T. Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium* (2011).

[12] CONSORTIUM, C. C., ET AL. Mirror link. http://www.mirrorlink.com/, 2013.

[13] EKBERG, J.-E., ET AL. Mobile trusted module (mtm)–an introduction.

[14] FOSTER, I., PRUDHOMME, A., KOSCHER, K., AND SAVAGE, S. Fast and vulnerable: a story of telematic failures. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)* (2015).

[15] GRAND, J. Jtagulator: Assisted discovery of on-chip debug interfaces. In *21st DefCon Conference, Las Vegas* (2013).

[16] GREENBERG, A. https://www.wired.com/2015/07/patch-chrysler-vehicle-now-wireless-hacking-technique/, 2015.

[17] GREENBERG, A. GM Took 5 Years to Fix a Full-Takeover Hack in Millions of OnStar Cars. https://www.wired.com/2015/09/gm-took-5-years-fix-full-takeover-hack-millions-onstar-cars/, 2015.

[18] GREENBERG, A. This Gadget Hacks GM Cars to Locate, Unlock, and Start Them. https://www.wired.com/2015/07/gadget-hacks-gm-cars-locate-unlock-start/, 2016.

[19] HOPPE, T., AND DITTMAN, J. Sniffing/replay attacks on can buses: A simulated attack on the electric window lift classified using an adapted cert taxonomy. In *Proceedings of the 2nd Workshop on Embedded Systems Security (WESS)* (2007).

[20] HOPPE, T., KILTZ, S., AND DITTMANN, J. Security threats to automotive can networks - practical examples and selected short-term countermeasures. *Reliability Engineering & System Safety 96*, 1 (2011), 11–25.

[21] ISHTIAQ ROUFA, R. M., MUSTAFAA, H., TRAVIS TAYLORA, S. O., XUA, W., GRUTESERB, M., TRAPPEB, W., AND SESKARB, I. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In *19th USENIX Security Symposium, Washington DC* (2010), pp. 11–13.

[22] KOSCHER, K., CZESKIS, A., ROESNER, F., PATEL, S., KOHNO, T., CHECKOWAY, S., MCCOY, D., KANTOR, B., ANDERSON, D., SHACHAM, H., ET AL. Experimental security analysis of a modern automobile. In *IEEE Symposium on Security and Privacy (SP)* (2010), IEEE, pp. 447–462.

[23] KOSTIAINEN, K., AND ASOKAN, N. Mechanisms for certificate revocation status verification on constrained devices, Dec. 19 2013. US Patent App. 13/910,613.

[24] LAURENDEAU, M. A beagleboard xm-based usb sniffer.

[25] MILLER, C., AND VALASEK, C. Adventures in automotive networks and control units. http://illmatics.com/car_hacking.pdf, 2013.

[26] MILLER, C., AND VALASEK, C. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA* (2015).

[27] NEWCOMB, D. Ford, gm open their dashboards to outside developers. www.wired.com/2013/01/ces-2013-ford-gm-app-developers/, 2013.

[28] NICOLAS BOICHAT, HUNYUE YAU, F. M. Google summer of code 2010 project.

[29] PARNO, B. Bootstrapping trust in a" trusted" platform. In *HotSec* (2008).

[30] ROEMER, R., BUCHANAN, E., SHACHAM, H., AND SAVAGE, S. Return-oriented programming: Systems, languages, and applications. *ACM Trans. Inf. Syst. Secur. 15*, 1 (Mar. 2012), 2:1–2:34.

[31] ROUSSEAU, S. http://www.popularmechanics.com/cars/a11735/gm-ford-to-developers-build-apps-for-our-cars-14954371/, 2013.

[32] SEACORD, R. C. *Secure Coding in C and C++*. Pearson Education, 2005.

[33] SERRA, J., RODRIGUES, J., ALMEIDA, T., AND MENDES, A. Multi-criticality Hypervisor for Automotive Domain. In *Inforum Conference* (2014).

[34] SOTIROV, A. Bypassing memory protections: The future of exploitation. In *USENIX Security* (2009).

[35] VERDULT, R., GARCIA, F. D., AND EGE, B. Dismantling megamos crypto: Wirelessly lockpicking a vehicle immobilizer. In *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)* (2015), pp. 703–718.

[36] WHITEHOUSE, O. Analysis of gs protections in windows vista, 2007.

[37] ZHANG, B. `http://www.businessinsider.com/volkswagens-cheating-engines-cant-be-easily-fixed-2015-9`, 2015.